**Artificially Intelligent Game Playing Agent which is able to predict the next best move**

import sys

import ast

class MiniMax(object) :

def \_\_init\_\_(self, profitableList={}, currentPlayerToPlay = 'r1'):

self.profitableList = profitableList

self.currentPlayerLastMove = '\*'

self.currentPlayerToPlay = currentPlayerToPlay

self.visited = []

self.playerScore = 0

self.opponentScore = 0

self.depth = 0;

self.finalValues = []

self.finalMove = ''

self.gameLength = len(self.profitableList)

def calculateHeuristicValues(self):

equationPart1 = sum(self.profitableList.values()) / self.gameLength

self.profitableList = dict((k, int(round(((v + equationPart1)/2.0)))) for k, v in self.profitableList.items())

def getOpponent(self, player):

return 'r1' if player == 'r2' else 'r2'

def updatePastPlayedStates(self, initialBoardState, expandedTerritories, player):

start = len(expandedTerritories) % 2

index = 0

if start != 0:

player = self.getOpponent(player)

while (index < len(exploredTerritories)):

if exploredTerritories[index] != 'pass' :

region = regionMatrixMapping.index(exploredTerritories[index])

for row in range(len(initialBoardState)):

if initialBoardState[row][region] == 1:

initialBoardState[row][region] = player

if initialBoardState[region][row] == 1:

initialBoardState[region][row] = player

if player == self.currentPlayerToPlay:

self.playerScore += self.profitableList.get(exploredTerritories[index])

self.currentPlayerLastMove = region

else :

self.opponentScore += self.profitableList.get(exploredTerritories[index])

self.visited.append(region)

index += 1

player = self.getOpponent(player)

self.currentPlayerLastMove

return initialBoardState

def getMovesForPlayer(self, board, player, lastMove, visited):

legitMoves = []

if lastMove == '\*' :

index = 0

while index < self.gameLength:

if index not in visited :

legitMoves.append(index)

index += 1

else :

for row in range(self.gameLength):

if board[lastMove][row] != self.getOpponent(player) and row not in visited and row != lastMove and board[lastMove][row] != 0:

legitMoves.append(row)

legitMoves.sort()

return legitMoves

def makeMove(self, move, board, player):

stateBoard = [row[:] for row in board]

for row in range(len(stateBoard)):

if player != self.getOpponent(player):

if stateBoard[row][move] == 1:

stateBoard[row][move] = player

if stateBoard[move][row] == 1:

stateBoard[move][row] = player

return stateBoard

def flipMinMax (self, minMax) :

return 'max' if minMax == 'min' else 'min'

def minimaxAlgorithm(self, board, player, lastMove, visited, playerScore, opponentScore, minMax, alpha, beta, depth):

moves = self.getMovesForPlayer(board, player, lastMove, visited)

if len(moves) < 1:

if player == self.currentPlayerToPlay:

if depth == 0 :

self.finalMove = 'PASS'

self.finalValues.append(int(round(playerScore)))

return {'PASS': playerScore}

elif len(self.getMovesForPlayer(board, self.getOpponent(player), visited[-1], visited)) > 0:

result = self.minimaxAlgorithm(board, self.getOpponent(player), visited[-1], visited, opponentScore, playerScore, self.flipMinMax(minMax), alpha\*1, beta\*1, depth - 1)

self.finalMove = 'PASS'

return result

else:

self.finalMove = 'PASS'

self.finalValues.append(int(round(playerScore)))

return {'PASS': playerScore}

else :

if depth == 0 :

self.finalValues.append(int(round(opponentScore)))

elif len(self.getMovesForPlayer(board, self.getOpponent(player), visited[-1], visited)) > 0:

return self.minimaxAlgorithm(board, self.getOpponent(player), visited[-1],visited, opponentScore, playerScore, self.flipMinMax(minMax), alpha\*1, beta\*1, depth - 1)

else :

self.finalValues.append(int(round(opponentScore)))

return {'PASS' : opponentScore}

else :

dict = {}

for move in moves:

stateBoard = [row[:] for row in board]

currentVisitedList = visited[:]

stateBoard = self.makeMove(move, stateBoard, player)

if len(currentVisitedList) > 0 :

preVisited = currentVisitedList[-1]

else :

preVisited = '\*'

opponentsPrevisit = '\*'

if len(currentVisitedList) > 0 :

opponentsPrevisit = currentVisitedList[-1]

currentVisitedList.append(move)

opponentMoves = self.getMovesForPlayer(stateBoard, self.getOpponent(player), opponentsPrevisit, visited)

if(depth == 0) :

if player == self.currentPlayerToPlay :

val = {regionMatrixMapping[move] : playerScore + self.profitableList.get(regionMatrixMapping[move])}

self.finalValues.append(int(round(playerScore + self.profitableList.get(regionMatrixMapping[move]))))

else:

val = {regionMatrixMapping[move]: opponentScore}

self.finalValues.append(opponentScore)

elif len(opponentMoves) < 1 and len(self.getMovesForPlayer(stateBoard, player, move, visited)) > 0 and (depth - 2) >=0:

val = self.minimaxAlgorithm(stateBoard, player, move, currentVisitedList, playerScore + self.profitableList.get(regionMatrixMapping[move]), opponentScore , minMax, alpha, beta, depth - 2)

else :

val = self.minimaxAlgorithm(stateBoard, self.getOpponent(player), preVisited, currentVisitedList, opponentScore, playerScore + self.profitableList.get(regionMatrixMapping[move]), self.flipMinMax(minMax), alpha \* 1, beta \* 1, depth - 1)

score = val.get(list(val.keys())[0])

value = val.itervalues().next()

dict.update({regionMatrixMapping[move]: score})

if minMax == 'max' :

if value >= alpha:

alpha = value

else :

if value <= beta:

beta = value

if alpha >= beta:

break

if minMax == 'min':

valueAtLevel = (sorted(list(dict.values()))[0])

else :

valueAtLevel = (sorted(list(dict.values()), reverse=True)[0])

choice = {k:v for k, v in dict.items() if v == valueAtLevel}

self.finalMove = str.upper(sorted(choice.keys())[0])

return choice

inputFile = open(sys.argv[2], 'r')

outputFile = open('output.txt', 'w+')

freshness = inputFile.readline().strip().lower();

playerToPlay = inputFile.readline().strip().lower();

profitabilityList = inputFile.readline().strip().split("),(");

regionMatrixMapping = [w.strip('()').lower().split(',')[0] for w in profitabilityList]

profitabilityList = dict(w.strip('()').lower().split(',') for w in profitabilityList)

profitabilityList = dict((k,eval(v)) for k,v in profitabilityList.items())

initialStateOfBoard = []

lengthOfProfitabilityList = len(profitabilityList)

count = 0;

while count < lengthOfProfitabilityList:

initialStateOfBoard.append(ast.literal\_eval(inputFile.readline().strip()))

count += 1

exploredTerritories = inputFile.readline().strip().lower().split(',')

depth = int(inputFile.readline().strip())

game = MiniMax(profitableList=profitabilityList, currentPlayerToPlay= playerToPlay)

if freshness == 'yesterday':

game.calculateHeuristicValues()

if '\*' not in exploredTerritories:

initialStateOfBoard= game.updatePastPlayedStates(initialStateOfBoard, exploredTerritories, playerToPlay)

currentPlayerLastMove = game.currentPlayerLastMove

else :

currentPlayerLastMove = '\*'

currentVisited = game.visited

alpha = float('-inf')

beta = float('inf')

depthToStart = depth-len(currentVisited)-exploredTerritories.count("pass")

if depthToStart >= 0 :

game.minimaxAlgorithm(initialStateOfBoard, playerToPlay, currentPlayerLastMove, currentVisited, game.playerScore, game.opponentScore, 'max', alpha\*1 , beta\*1, depthToStart)

outputFile.write(game.finalMove+'\n')

outputFile.write(','.join([str(i) for i in game.finalValues]))

inputFile.close()

outputFile.close()